Neural Joint Space Implicit Signed Distance Functions for Reactive Robot Manipulator Control

Mikhail Koptev, *Graduate Student Member, IEEE*, Nadia Figueroa, *Member, IEEE*, and Aude Billard, *Fellow, IEEE*

***Abstract*—In this letter, we present an approach for learning a neural implicit signed distance function expressed in joint space coordinates, that efficiently computes distance-to-collisions for arbitrary robotic manipulator configurations. Computing such distances is a long standing problem in robotics as approximate representations of the robot and environment geometry can lead to overly conservative constraints, numerical instabilities and expensive computations – limiting real-time reactive control and task success. Leveraging GPU parallelization and the differentiable nature of the proposed distance function allows for fast calculation of gradients with respect to the neural network inputs, providing a continuous repulsive vector field directly in joint space. We show that the learned high-resolution collision representation can be used to achieve real-time reactive control by i) formulating it as a collision-avoidance constraint for a quadratic programming (QP) inverse kinematics (IK), and ii) introducing it as a collision cost in a sampling-based joint space model predictive controller (MPC). For a reaching benchmark task with a 7DoF robot and dynamic obstacles intentionally obstructing the robot's path we achieve average 250 Hz control frequency with QP-IK and 92 Hz with MPC, showing an accelerated performance of 15% for QP-IK and 40% for MPC over baseline distance computation techniques.**

***Index Terms*—Collision Avoidance, Reactive and Sensor-Based Planning, Model Learning for Control.**

I. Introduction

MOTION planning is a crucial element of robotic control and has continuously attracted the interest of researchers throughout the years. Industrial, retail, and lately, domestic scenarios require robotic motion to satisfy certain constraints, such as energy minimization, smooth trajectories, goal-reaching, and orientation keeping. Yet, the most important constraint to satisfy in any motion planning task is to avoid collisions and self-collisions. Standard techniques avoid such collisions by computing distances between geometric approximations of the robot's body segments represented as spheres, ellipsoids, swept volumes and even meshes [1]. One can then formulate constraints or costs for the motion planning algorithm of choice based on the computed minimum distances [2].

While avoiding self-collisions is important, handling external collisions, i.e., collisions between the robot and its environment, is essential for a robot that must operate in constrained and dynamic environments. These collisions can be categorized into two types: collisions with static obstacles, such as furniture or walls, and collisions with moving obstacles, including other robots or humans. Avoiding moving obstacles is crucial for consumer technology, where robots should operate in an environment shared with humans, which assumes that the robot must always be prepared to avoid humans in its workspace to prevent possible injuries. With the advent of compliant collaborative robots equipped with force/torque sensors and robust vision systems, a robot can halt its motion when a collision is detected [3]. Yet, advanced collaborative systems are expected to continue task execution while avoiding any type of obstacles in both a preemptive and reactive manner. Additionally, certain scenarios in human-robot interaction require maintaining direct contact between the robot and humans [4], [5]. Thus, it is essential to have a notion of a repulsive vector field and corresponding tangential plane to handle the contact. Having those defined directly in joint space helps to plan more efficiently.

Previous works [6], [7] demonstrated that self-collisions of a redundant system could be represented as a static boundary in the high-dimensional joint space of a robot. When this boundary is approximated as a continuously differentiable function of class $\mathcal{C}^1$, the gradients with respect to the input robot configuration essentially represent a repulsive vector field directly in the joint space of the robot; that repulsion can be used as a constraint in a control optimization routine. In this letter, this idea is extended to account for collisions with obstacles in the robot's workspace. We note that static self-collision boundaries in joint space can be interpreted as implicit signed distance-to-collision functions, if the robot state space is complemented with the Euclidean 3D space. We present a novel approach for constructing an implicit distance function for collision evaluation between a robot in arbitrary configurations and any point in the three-dimensional workspace of the robot. The learned neural model allows for efficient and highly-parallelizable batched distance and gradient queries via GPU.

The problem of collision avoidance is generally treated as a constraint in a path-planning routine [8]. Optimization and collision checking computations are unfeasible to perform in real-time, so optimal trajectories are computed offline before execution. However, if the obstacles near the robot are moving unpredictably, the control must be *reactive* and adjust the trajectory at execution time.

Manuscript received 3 July 2022; accepted 1 December 2022. Date of publication 8 December 2022; date of current version 16 December 2022. This letter was recommended for publication by Associate Editor J. Godoy and Editor A. Bera upon evaluation of the reviewers' comments. This work was supported by European Comission SAHR, under Grant 741945. *(Corresponding author: Mikhail Koptev.)*

Mikhail Koptev and Aude Billard are with the École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland (e-mail: mikhail.koptev@epfl.ch; aude.billard@epfl.ch).

Nadia Figueroa is with the University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: nadiafig@seas.upenn.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3227860, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3227860

In this letter, we assume the obstacles to be dynamic and, hence, investigate the applicability of the learned distance function to achieve real-time performance with two state-of-the-art *reactive* control methods: i) a collision-aware QP (quadratic program)-IK (inverse kinematics) solver that optimizes for the next immediate step and has shown real-time performance for high-dimensional humanoid robots [6], [9], and ii) a sampling-based MPC that has a short look-ahead horizon [10], [11]. Both methods are reported to generate solutions with at least 100 Hz frequency, which should allow real-time reactive obstacle avoidance.

Paper organization: In Section II we summarize related works for collision distance computation and describe their usage and applicability to *reactive* motion planning approaches. Section III presents the mathematical problem formulation, including assumptions, definitions and goals of this letter. In Section IV we present our proposed approach for learning a neural joint space implicit signed distance function (Neural-JSDF) for a given robot. Subsequently, Section V covers the application of the proposed Neural-JSDF to a QP-IK solver and sampling-based MPC approach for reactive control. Finally, we evaluate the performance of these two reactive control techniques in simulation and with real robotic experiments in Section VI.

II. Related Works

Collision avoidance between the robot and arbitrary objects in its workspace, including links of the same robot, requires having the notion of distance between the robot body and the obstacles. For situations when a robot operates in a static environment, it is possible to pre-compute the Euclidean Distance Transform (EDT) and acquire the corresponding collision map. However, to account for moving obstacles, EDT must be adjusted on every change of obstacle position, which is unfeasible from a computational perspective [8]. Another approach to handle collisions is via proprioceptive sensors, which could include monitoring electrical current in robot's drives or comparing measured torques with the control law [3]. Such methods help detect contact reliably but can not be used in path-planning or reactive control scheme when we seek to preemptively avoid collisions. State-of-the-art distance computation approaches seek to find a trade-off between accuracy and computation efficiency in order to be used for fast real-time queries.

The most straightforward way to calculate the distance between the robot's body and an obstacle would be to calculate the minimal Euclidean distance between meshes of the robot and the object in question. However, from a computational perspective, direct collision queries comparing the distance between each pair of robot-obstacle points are not feasible, as precise meshes of robot and environment objects may contain hundreds of thousands of vertices. Typically the geometric representation of colliding bodies is simplified with bounded volumes, convex hulls, or even simple spherical capsules, allowing for a trade-off between exact representation and collision query speed [12], [13]. Further, a vector connecting the closest points can be treated as a repulsion in task space and translated to the joint space using Jacobian transpose projection or formulating the task-space repulsion as a cost in optimization-based control schemes [14], [15]. Nevertheless, it has been shown that such geometric primitive approximations may lead to overly-conservative constraints that can interfere with the task or lead the robot to a collision, numerical instabilities and an increase in computational complexity as the number of obstacles increases; limiting their use for real-time reactive control [16], [17], [18]. To alleviate some of these issues, there has been a recent spike in works to account for collisions directly in the joint space of a robot [19], [20], [21]. This addresses some issues with kinematic-based collision detection, such as IK-caused singularities and local minima. Additionally, having the distance expressed in joint space presents an alternative to task-space approximation with geometric primitives, as it becomes possible to compute repulsion directions directly in the joint space of the robot, which allows to use the tangent plane in joint space.

With the advent of parallelization for ML techniques, several approaches have been proposed to learn those distance functions in hopes to achieve and boost real-time performance. For static scenarios or self-collisions, there are works to represent the collision boundary as a Support Vector Machine (SVM), or Neural Network (NN) [6], [7], [22] learned by sampling the joint space of the robot. Notably, several works rely on active learning and online sampling to adjust the model learned for a static environment to a dynamic one [19]. It is further extended to use the gradients of the learned model in various path-planning pipelines [20]. However, even for a low-DoF manipulator, the reported update step (0.27 seconds) is too large to be used in real-time. A NN provides seamless batching and parallelization that significantly speeds up collision detection. In [23], authors present a trained model that can predict the occupancy grid using scene point-cloud and joint configuration of a robot as inputs. The learned model provides high collision detection rate and good performance in case of batched queries. However, the model is reported to be unsuitable for real-time control due to slow point-cloud processing. The authors of [21] learn collisions as a function of the combined robot state and parametrized environment state. They further show its application in parallelized rapidly-exploring random tree planner (RRT), proving that such a method can be more effective than traditional collision checking for batched queries. Additionally, the authors leverage the differentiability of the model to adjust the path planning. Our work can be seen as a further abstraction of this method for arbitrary obstacles in an unstructured environment, with application to real-time control algorithms.

III. Problem Formulation

We consider a robotic manipulator with m degrees of freedom and K links, whose state is described by joint angles $q = [q^1, \ldots, q^m] \in \mathcal{C}$. All joints of the robot are revolute type and bounded by joint-limits $q \in [q_{\min}, q_{\max}]$, thus $q \in \mathcal{C} \subset \mathbb{R}^m$.

In this work, we seek to control a robotic manipulator in joint space to reach a task-space goal $x^* \in \mathcal{SE}(3)$ for its end-effector that could be pre-defined or generated by a task-space control law, all the while handling collisions between the robot's body and static and dynamic objects in real-time; i.e., with a desired control loop frequency of $\geq$100 Hz.

A. Assumptions & Definitions

Let us assume $\mathcal{B} \subset \mathbb{R}^3$ to be the set of points that geometrically describe the physical body of the robot. The points of the k-th link of the robot create subsets $\mathcal{B}_k \subset \mathcal{B}$, $k = 1, \ldots, K$. The robot's kinematics is known, with forward kinematics (FK) denoted by $f : \mathcal{C} \times \mathcal{B} \to \mathbb{R}^3$, mapping a robot configuration $q \in \mathcal{C}$ and arbitrary robot body point $x \in \mathcal{B}$ to a workspace point $f(q, x) \in \mathbb{R}^3$. We can define minimal distances between the k-th

link of the robot and arbitrary point $y \in \mathbb{R}^3$ in the workspace:

$$d_k(q, y) = \min_{x \in \mathcal{B}_k} ||f(q, x) - y||. \tag{1}$$

The workspace of the robot contains static and dynamic obstacles described by the sets of points $\mathcal{O}^s \subset \mathbb{R}^3$ and $\mathcal{O}^d \subset \mathbb{R}^3$, respectively. Dynamic obstacles can be generalized and include static, such that $\mathcal{O}^s \subset \mathcal{O}^d$, hence we will herein refer to the obstacle set as $\mathcal{O} \subset \mathbb{R}^3$ regardless of obstacles being static or dynamic. This definition allows us to write down minimal distances between the robot in configuration q and the environment as:

$$d_{\min}(q) = \min_{k,y} d_k(q, y), \tag{2}$$

where $y \in \mathcal{O}$ are points belonging to obstacles in the workspace. Naturally, if we seek to avoid collisions between the robot and the environment with a given margin l, any command position q must satisfy the condition $d_{\min}(q) > l$.

B. Goals

Two main goals are defined to achieve our objective:

- The *first goal* of this letter is to learn a regression function $\Gamma(q, y)$ to approximate a K-dimensional minimal distance vector $d(q, y) = [d_1(q, y), \ldots, d_K(q, y)]$ with each entry representing the minimal distance between the k-th link and and arbitrary 3D point. Function $\Gamma(q, y)$ represents a distance field in the robot's workspace depending on the joint configuration of the robot. Additionally, $\frac{\partial \Gamma(q,y)}{\partial q}$ is a vector field defined in the joint space of the robot, providing information on the direction to (and away from) potential collisions. We provide the theoretical derivation and properties of the proposed distance field $\Gamma(q, y)$ as well as the learning architecture and evaluation in Section IV.
- The *second goal* is to apply the learned function $\Gamma(q, y)$ and its gradient to enhance reactive control methods, by i) formulating a collision-avoidance constraint in a QP-based IK controller [6], and ii) formulating a collision-avoidance cost to leverage the parallelization properties in a sampling-based MPC approach [11]. Corresponding method formulations are described in Section V. Real-time experiments reported in Section VI.

IV. Learning the Neural Joint Space Implicit Signed Distance Function

This section expands ideas from our previous work [6] and [7] to allow for distance evaluation between the robot and an arbitrary point in the robot's three-dimensional workspace, along with demonstrating how to construct the corresponding implicit signed distance field.

A. Implicit Signed Distance Field

Let us consider the expanded state-space $\mathbb{R}^m \times \mathbb{R}^3$, consisting of the robot state $q \in \mathbb{R}^m$ and a Euclidean point $y \in \mathbb{R}^3$ in the robot's workspace. For each expanded state, a unique minimal distance exists between the robot in configuration q and the 3D point at position y. Hence, a static implicit distance field function exists in this space.

We propose to build a neural representation $\Gamma(q, y) : \mathbb{R}^m \times \mathbb{R}^3 \to \mathbb{R}^K$, by learning the minimal distances between the robot's links and arbitrary points in the workspace. For typical redundant robotic manipulators $m < K$. For example, the

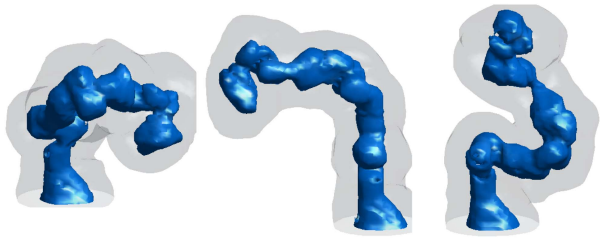

Fig. 1. Illustration of the learned implicit distance isosurfaces $\Gamma(q, y) = 0$ cm (solid) and $\Gamma(q, y) = 10$ cm (transparent) for various configurations of the 7 DoF (Degrees-of-Freedom) Franka Emika Panda robot.

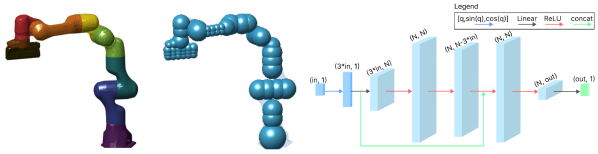

Fig. 2. (left) Mesh representation of the Franka Emika, coloured segments represent the $K = 9$ links, and (center) sphere approximation (with $S = 55$) of the robot geometry, (right) NN Architecture for the $\Gamma(q, y)$ function, featuring positional encoding and skip-connection option.

7 DoF Franka Emika Panda robotic manipulator can be defined with $m = 7$ and $K = 9$ (excluding the gripper fingers from the model), see Fig. 2. The resulting function $\Gamma(q, y) = [\Gamma_1, \ldots, \Gamma_K]$ should represent minimal distances $d(q, y) = [d_1(q, y), \ldots, d_K(q, y)]$ between the point and each robot link. The partial derivative of the vector-valued distance field $\Gamma(q, y)$ with respect to q results in the following Jacobian matrix, $\text{Jac}_q(\Gamma(q, y)) \in \mathbb{R}^{m \times K}$,

$$\text{Jac}_q\left(\Gamma(q, y)\right) = \frac{\partial \Gamma(q, y)}{\partial q} = \begin{bmatrix} \frac{\partial \Gamma_1(q,y)}{\partial q_1} & \cdots & \frac{\partial \Gamma_K(q,y)}{\partial q_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Gamma_1(q,y)}{\partial q_m} & \cdots & \frac{\partial \Gamma_K(q,y)}{\partial q_m} \end{bmatrix} \tag{3}$$

where each k-th column vector of $\text{Jac}_q(\Gamma(q, y))$ corresponds to the gradient of each k-th scalar distance function $\Gamma_k(q, y)$ with respect to q; i.e., $\text{Jac}_q(\Gamma(q, y)) = [\nabla_q \Gamma_1(q, y), \ldots, \nabla_q \Gamma_K(q, y)]$ and $\nabla_q \Gamma_k(q, y) \in \mathbb{R}^m$ as,

$$\nabla_q \Gamma_k(q, y) = \frac{\partial \Gamma_k(q, y)}{\partial q} = \sum_{i=1}^{m} \frac{\partial \Gamma_k(q, y)}{\partial q_i} \hat{q}_i, \tag{4}$$

with $\hat{q}_i$ as the unit vector indicating the i-th dimension. Such gradient functions help explain how each joint influences the distance-to-collision of obstacle y with the k-th link. Each k-th distance function, $\Gamma_k(q, y)$, can be considered an admissible potential field for obstacle avoidance [24], [25] in joint space. Consequently, $\nabla_q \Gamma_k(q, y)$ becomes a repulsive joint space vector field, which is helpful for control [26].

Therefore, learning $\Gamma(q, y)$ will enable the evaluation of distances between the robot and points on the obstacles and using $\nabla_q \Gamma(q, y)$ to represent the repulsive vector field. Ideally, $\Gamma(q, y) = [d_1(q, y), \ldots, d_K(q, y)]$ for any combination of state q within joint-limits and point y in the robot's workspace. A Multi-Layer Perceptron (MLP) is used to learn $\Gamma(q, y)$. The data collection procedure and network architecture are discussed in the following subsections.

B. Dataset Generation

Given the exact geometry of the robot's body (for example with 3D CAD models or URDFs) allows for the collection of a synthetic dataset of exact distance values $d_k(q, y)$, as in (1), for various q and y at training time. Each sample contains the robot state q, workspace point y, and target vector $d = [d_1, \ldots, d_K]$ consisting of minimal distances between links of the robot and point y. Similar to [6], we perform a uniform random sampling within the joint limits of the robot to generate the training dataset. The joint limits for training data are expanded by 5% in both directions to aid generalization for predictions close to joint limits. For each sampled robot configuration q, 1000 various workspace points y are collected. The final dataset contains five million entries, where 50% of configurations are collided or close to collision, meaning that $\exists k : d_k(q, y) \leq 1$ cm, and 50% are configurations with a minimal distance exceeding 1 cm, such that $\forall k, d_k(q, y) > 1$ cm. Such balancing allows for a better approximation of the null iso-surface, which represents the exact geometry and surface of the robot's body. The collided half of the dataset is balanced to include collisions for links in equal proportions. Non-colliding data points are distributed uniformly between 1 cm and 100 cm from collision. Another set of one million data points with non-modified joint limits are collected for testing.

Implementation: We compute distances between points and triangulated surfaces with code by [27] in combination with the fast collision checking library (FCL) [28] to calculate the signed distance between the mesh of the robot and the point in the workspace. The data collection procedure takes 90 minutes on a 12-core 3.7 GHz CPU.

C. Network Architecture

Various network architecture choices were investigated to achieve the optimal performance. A simple fully-connected MLP was used as the baseline. The rectified linear activation function (ReLU) is used to provide faster forward and backward passes, and the root mean square error (RMSE) is used as the loss function.

The function $\Gamma(q, y)$ implicitly learns the robot's forward kinematics; thus, it might be helpful to build a feature vector as a concatenation of joint angles and their corresponding sine and cosine values: $q_{in} = [q, \cos(q), \sin(q)]$. Similar *positional encoding (p.e.)* is discussed in [29], and the authors conclude that it helps with learning of high-frequency functions. Another perspective on the positional encoding is that it introduces non-linear features frequently appearing in analytic FK equations, simplifying the regression task for NNs. To choose the optimal network parameters (number of layers and their size) for the MLP with positional encoding feature vector we performed a grid-search training with different depths, D, and layer sizes, N. For each pair of N and D, the network was trained on 2 M training data points with different initial weights five times. The resulting average RMSE and its standard deviation are reported for 200 k testing data points in Table I. As can be seen, increasing the value of N improves the regression accuracy; however, as the network size is proportional to $\mathcal{O}(N^2 D)$ complexity, the smaller value $N = 256$ is chosen. For larger values of D, within the same amount of epochs, MLP fails to improve upon shallower networks (for $N > 128$). The chosen architecture used had $D = 5$ and $N = 256$. The behavior of the RSME as we increase the depth of the networks in Table I suggests that we have a degradation problem; i.e., either we are over-fitting or we have vanishing and/or exploding gradients. One way to alleviate this is to introduce *skip-connections (s.c.)* between the input features and deeper layers of the network, essentially convexifying the training loss function. Apart from combating the vanishing/exploding gradients problem, we hypothesized that reintroducing the trigonometrical input features to deeper layers could yield better FK approximation. An example of the network with skip-connection between input and the fourth layer is provided in Fig. 2. Hence, we performed a further analysis of the MLP with and without p.e. and s.c. as shown in Table II. Each architecture type is trained ten times, and the resulting RMSE distributions are studied by means of two-sample t-test. Each distribution is compared with the basic MLP (as in first row of Table II), and resulting p-value is reported. The p-value indicates the likelihood of two distributions to be the same ($p = 1$) or to be different from each other ($p = 0$). For skip-connection $p = 0.93$, indicating that this feature is unlikely to improve the learning. The table shows that positional encoding leads to better distance approximation, while skip-connection does not bring significant improvements. Based on these findings, a simple MLP with $D = 5$ and $N = 256$ along with a positional encoding for the regression task is used. This network is similar to Fig. 2 but without the skip-connection. There, $in = 10$, with 7 for DoF and 3 for the workspace point position, and $out = 9$ representing d_k for $k = 1..9$ links.

TABLE I
A COMPARISON OF VARIOUS LAYER SIZES N AND HIDDEN LAYERS AMOUNT D

N/D	**3**	**5**	**7**	**9**
64	4.88 (3.61)	3.63 (2.67)	3.50 (2.48)	**3.23 (2.22)**
128	3.43 (2.53)	2.38 (1.75)	**2.33 (1.62)**	2.47 (1.67)
256	2.74 (2.09)	**1.89 (1.38)**	2.07 (1.50)	2.07 (1.40)
512	2.71 (2.06)	**1.71 (1.23)**	1.77 (1.27)	1.96 (1.34)

Each architecture is trained on the same data for 10,000 epochs. The RMSE and standard deviation averaged between all links (and across five training instances) are provided for the 200 k testing dataset.

TABLE II
A COMPARISON OF NETWORK ARCHITECTURES

Architecture	**Mean RMSE, cm**	**p-value**
MLP	2.25 (1.66)	1.0000
MLP *(s.c.)*	2.28 (1.65)	0.9331
MLP *(p.e.)*	**1.89 (1.38)**	0.0002
MLP *(s.c + p.e.)*	1.93 (1.43)	0.0013

Each trained on the same data for 10,000 epochs. The network parameters are $L = 5$ and $N = 256$. The results are averaged between ten trainings. *P*-value indicates the result of two-sample t-test run against first row of the table.

D. Learning Results

After selecting the hyperparameters, the NN was trained for 100,000 epochs, taking approximately 8 h on RTX3090.[1]

Accuracy: Averaging between all links, $\Gamma(q, y)$ predicts minimum distances with an RMSE of 1.05 cm and a standard deviation of 0.81 cm. While the problem is posed as a regression, the ability of the learned model to distinguish between collided and free configurations, i.e., accuracy of predicting $\text{sign}(\Gamma(q, y))$,

[1]The source code and supplementary video is available at `https://github.com/epfl-lasa/Neural-JSDF`

TABLE III
COMPUTATIONAL PERFORMANCE OF THE LEARNED FUNCTION

Batch size	CPU		GPU	
	Batch time, μs	Sample time, μs	Batch time, μs	Sample time, μs
1	32.141	32.141	74.340	74.340
10	115.998	11.600	79.148	7.915
100	426.743	4.267	98.689	0.987
1000	2341.552	2.342	103.147	0.103
10000	22734.501	2.273	281.336	0.028
100000	314058.185	3.141	2467.827	0.025

All results are averaged on 10 k runs with 12-core 3.7 GHZ CPU and RTX3090 GPU.

TABLE IV
PERFORMANCE OF LEARNED FUNCTION $\Gamma(q, y)$ IN TERMS OF RMSE AND ITS STANDARD DEVIATION FOR CLOSE QUERY POINTS WITH $d_k \leq 10$ CM, AND FAR ONES WITH $d_k > 10$ CM, AND IN TERMS OF THE ACCURACY a_{coll} OF PREDICTING COLLISIONS, REPRESENTED AS SIGN$(\Gamma(q, y))$ FOR CONFIGURATIONS WITH $d_{\min} < 3$ CM.

Link, k	a_{coll}	Close RMSE, cm	Far RMSE, cm
1	0.98	0.37 (0.24)	0.40 (0.27)
2	0.99	0.32 (0.20)	0.43 (0.28)
3	0.98	0.38 (0.24)	0.46 (0.30)
4	0.97	0.56 (0.37)	0.57 (0.39)
5	0.93	0.73 (0.51)	0.68 (0.46)
6	0.88	1.02 (0.69)	1.06 (0.74)
7	0.87	1.29 (0.85)	1.25 (0.86)
8	0.83	1.40 (0.89)	1.46 (1.00)
9	0.76	1.83 (1.17)	2.06 (1.37)
Avg	0.90	1.04 (0.78)	1.06 (0.82)

was also investigated for configurations with $d_{\min} < 3$ cm. For such points, the classification accuracy of sign$(\Gamma(q, y))$ is 0.90 when averaged between links. The performance of $\Gamma(q, y)$ in terms of RMSE and its standard deviation for each robot link is reported in Table IV. Fig. 1 shows iso-surfaces for different values of the learned $\Gamma(q, y)$.

Computation Time: The computational performance of the learned function is investigated, and the results are presented in Table III. It shows that batching improves the per-sample performance, which is expected with NN. Additionally, it shows that for a single sample, computations are more efficient on CPU. For reference, authors of [18] report that one distance query between two convex hulls takes 0.6 μs using standard Gilbert-Johnson-Keerthi (GJK) algorithm [1]. The learned network calculates distances between $K = 9$ links and a single sphere, i.e., equivalent to performing nine such queries. Thus, if we assume that GJK is not parallelized, we can use 5.4 μs as a baseline for a single robot-obstacle distance evaluation. If the control algorithm can benefit from batched distance queries (e.g. it is sampling based, or there are multiple obstacles), the NN can be up to 1000x faster than the standard GJK algorithm used to compute distances.

V. REACTIVE CONTROL WITH NEURAL-JSDF

As described previously, the learned function $\Gamma(q, y)$ approximates the distances between the robot's links in a given joint configuration, q, and Euclidean point, y, in its workspace. To treat collisions with moving, and possibly deforming obstacles of non-fixed shape and size we approximate the obstacle shape with $s = 1..S$ spheres with centers y_s and radii r_s. Note, point-cloud obstacle representations fit naturally in such framework, with $r_s = 0$, $\forall s$.

This section presents two approaches, i) QP-IK and ii) Sampling-based MPC, leveraging on various properties of the learned Neural-JSDF $\Gamma(q, y)$, such as repulsive gradient in joint space, and efficient batch input processing.

A. Reactive Collision-Avoidance IK

Similar to [6], the learned function $\Gamma(q, y)$ can be used to formulate a constraint in a QP-IK solver:

$$\min_{\Delta q, \delta} \delta^T Q \delta + \Delta q^T R \Delta q$$

$$\text{s.t.} \quad \begin{cases} f(q) + \dfrac{\partial f(q)}{\partial q} \Delta q = x + \delta \\ q_i^- < q_i + \Delta q_i < q_i^+, \quad i = 1..m \\ -\nabla_q \Gamma_k(q, y_s)^T \Delta q \leq \ln(\Gamma_k(q, y_s) - r_s), \quad \forall k, s. \end{cases} \tag{5}$$

In (5), the goal is to minimize joint displacement Δq with damping term R, to satisfy the kinematic constraints given by forward kinematics $f(q)$, Jacobian $\frac{\partial f(q)}{\partial q}$ and cartesian task $x \in \mathcal{SE}(3)$. These tasks may specify desired positions and orientations for chosen links, but further we assume they only contain tasks for the end-effector. Additionally, for situations where the solver fails to satisfy the reaching constraint, slacks δ are introduced and minimized with corresponding weights Q. The second constraint accounts for joint-limits, which are defined as q_i^- and q_i^+ for $i = 1..m$ joints.

Finally, the last line in (5) defines the proposed collision avoidance behaviour with the learned implicit signed distance function. It represents $K \times S$ constraints, one for each pair of $k = 1..K$ links and $s = 1..S$ spherical obstacles. $\Gamma_k(q, y_s)$ is the k-th component of the implicit distance function output vector, for $k = 1..K$ links. Finally, y_s are centers of spherical obstacles $s = 1..S$ with radii r_s. When the robot is far from collisions and $\Gamma_k(q, y_s) > 0, \forall k$, the set of constraints is relaxed. If the k-th link is close to the s-th obstacle, then $\ln(\Gamma_k(q, y_s) - r_s)$ becomes negative, and the IK solver is forced to align joint motion Δq with the corresponding gradient $\nabla_q \Gamma_k(q, y_s)$, repelling the robot away from the collision boundary and satisfying collision avoidance constraint.

B. Sampling-Based Model Predictive Control

Additionally, we propose to use the learned Neural-JSDF in a more complex algorithm with a look-ahead horizon to address the local nature of the QP controller described in the previous section. Recent work [11] demonstrates sampling-based MPC for robotic manipulators leveraging on massive parallelization via a GPU to speed up FK and cost function computations. The learned function $\Gamma(q, y)$ fits naturally in such framework allowing for efficient batched collision checking. Below, the basics of Model Predictive Path Integral (MPPI) control are introduced. For a discrete-time system at time t, the robot is controlled by a joint space acceleration command u_t. This command is sampled from a policy $\pi_t = \Pi_{h=1}^H \pi_{t,h}$, where H is a look-ahead horizon, and policies $\pi_{t,h}$ are simple Gaussians defined by means $\mu_{t,1}, \ldots, \mu_{t,H}$ and covariances $\Sigma_{t,1}, \ldots, \Sigma_{t,H}$. At every iteration the sampling-based MPC algorithm, proposed in [11], samples a batch $\{u_{i,h}\}_{i=1..N}^{h=1..H}$ of N control sequences of length H from current distribution π_t. After that, the roll-out states $\{x_{i,h}\}_{i=1..N}^{h=1..H}$ are computed using an approximate dynamics function. The corresponding costs, $\{c_{i,h}\}_{i=1..N}^{h=1..H}$, are calculated as a weighted sum of goal-reaching, joint-limit avoidance,

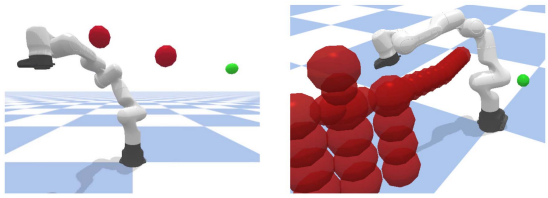

Fig. 3. Benchmark scenarios for the reaching task. Goal is depicted as a green sphere, obstacles are represented with red spheres. (left) Scenario A, where spheres placement is different across the experiments, (right) Scenario B, human shape approximated with 30 spheres, with variable arm placement.

contingency stopping, and self- and environmental-collision avoidance costs.

The means and covariances of the Gaussian policies, $\pi_{t,h} = \mathcal{N}(u_{t,h}|\mu_{t,h}, \Sigma_{t,h})$, are then updated using a sample-based gradient of a risk-seeking objective function. The update rule for $\mu_{t,h}$ is:

$$\mu_{t,h} = (1-\alpha_\mu)\mu_{t-1,h} + \alpha_\mu \frac{\sum_{i=1}^{N} w_i u_{i,h}}{\sum_{i=1}^{N} w_i} \tag{6}$$

where α_μ is a filtering coefficient, and

$$w_i = \exp \frac{-1}{\beta} \left(\sum_{h=1}^{H-1} \gamma^h c(x_{i,h}, u_{i,h}) + \gamma^H \hat{c}(x_{i,H}, u_{i,H}) \right) \tag{7}$$

is a cost-based weighting coefficient for sampled trajectories. In (7), $c(x,u)$ is a task-specific cost, $\hat{c}(x,u)$ is the terminal cost, and $\gamma \in [0,1]$ is a discount factor to balance between immediate rewards and final goal. We refer the reader to [10], [11] for a detailed derivation of the MPPI algorithm. The distinctive property of MPPI is that it can handle a wide range of cost formulations, as follows [11]:

$$\begin{aligned} c(x,u) = \alpha_p c_{\text{pose}} + \alpha_s c_{\text{stop}} + \alpha_j c_{\text{joint}} \\ + \alpha_m c_{\text{manip}} + \alpha_c (c_{\text{coll}} + c_{\text{self-coll}}), \end{aligned} \tag{8}$$

where goal-reaching, contingency stopping, joint limit avoidance, manipulability and collision costs are combined with respective weights. In (8), the collision cost, c_{coll}, is discrete, yielding $c_{\text{coll}} = 1$ if the robot is in collision with the environment, and $c_{\text{coll}} = 0$ otherwise. We use $\text{sign}(\Gamma(q,y))$ to efficiently perform batched collision queries for computing the cost function.

VI. Evaluation in Simulation and on Real Robot

A. Evaluation Scenarios and Metrics

We evaluate the performance of the reactive control methods introduced in the previous section (QP-IK and MPPI) using the learned Neural-JSDF for collision avoidance in two scenarios of different complexity. In both cases, the robot's goal is to perform a reaching motion from initial configuration, to the goal defined in the task space.

Scenarios: Scenario (A) includes two disjoint spherical obstacles in front of the robot. Scenario (B) imitates human presence in the robot's workspace (refer to Fig. 3). Human body is approximated with 30 spheres of varying radii, fitting the torso, head, and right arm. Both control methods are expected to work with at least 50 Hz frequency, providing reactive behavior for avoiding moving obstacles. However, as dynamic information (such as obstacles velocity) is not directly incorporated into avoidance constraints, these methods are tested in a quasi-static environment. Additionally, that helps to achieve reproducible results. The benchmark consists of 100 experiments for randomized positions of collision obstacles. For scenario A the vertical placement of the obstacles differs between the experiments, while scenario B simulates the human with different arm positions, obstructing the robot's workspace. The robot's initial configuration and goal position are constant across the experiments.

TABLE V
Performance Comparison for Scenario A (Two Disjoint Spheres)

	QP(Sph)	QP(NN)	MPC(Sph)	MPC(NN)
Success rate, %	1.00	1.00	1.00	1.00
Reaching time, s	4.53	3.24	4.17	2.53
Clearance, cm	1.32	1.14	0.97	0.84
Frequency, Hz	217	249	66	92

TABLE VI
Performance Comparison for Scenario B (Human-Shaped Obstacle Approximated With 30 Spheres)

	QP(Sph)	QP(NN)	MPC(Sph)	MPC(NN)
Success rate, %	0.91	0.92	1.00	1.00
Reaching time, s	5.76	4.09	4.30	3.17
Clearance, cm	1.24	1.21	0.86	0.82
Frequency, Hz	204	220	49	68

Metrics: The results are provided in Tables V and VI. There, success rate indicates a number of experiments with successful goal reaching (i.e. no collisions and free from local minima), and clearance stands for average minimal distance-to-collision across successful trajectories. Both methods were tuned to behave conservatively and not allow for any collisions. Overall, QP is more reactive than MPC, but due to QP's local nature and lack of optimization horizon, it has a lower success rate in a more complex scenario.

B. Implementation Details and Discussion

1) Baseline Approach (Spheres Approximation): Links of the robot are approximated with 55 spheres of various radii as shown in Fig. 2. This number of spheres and their radii were chosen to best represent the actual geometry of the robot without being overly conservative. For simplicity, p_k and r_k are used to denote a sphere belonging to the $k-$th link of the robot. For each obstacle sphere with a center y_s, and a radius r_s, the closest sphere, (p_k, r_k), belonging to the robot can be found. The distance between the robot and the obstacle is then $d_{k,s} = ||y_s - p_k|| - r_s - r_k$, and $v_{k,s}$ is a vector connecting two spheres, that acts as a repulsion in task space. For the QP-IK, collision avoidance constraints in (5) are then replaced with the following:

$$(J_k^{rep}(q)\Delta q)^T v_{k,s}^{rep} \leq \ln(d_{k,s}), \tag{9}$$

where $J_{k,s}^{rep}(q)$ is a Jacobian for the corresponding point on the robot body, and $v_{k,s}^{rep}$ is scaled to match the magnitude of Δq. These constraints cover all pairs of links and obstacles, forcing the solver to generate motion in a tangential plane to the obstacles, when the distance to collision $d_{k,s}$ is small.

2) QP-IK: CVXGEN [30] is used to solve the described QP problem. The values of Γ and $\nabla_q \Gamma$ are evaluated on the 12-core 3.7 Ghz CPU and passed to the solver. We evaluate

Fig. 4. Snapshots of a reaching task with the human upper body as an obstacle approximated with a collection of 30 spheres. See *multimedia attachment*.

the performance of the proposed collision avoidance IK solver using the learned Neural-JSDF $\Gamma(q, y)$ compared to formulating the collision avoidance constraint with the traditional method to detect collisions. Our method efficiently combines the learned function $\Gamma(q, y)$ and its gradient to create the repulsive force around obstacles, showing a 15% increase of computation speed for the IK solver over using the baseline technique for collision detection. However, this approach is prone to local minima, especially in case of non-convex obstacles, demonstrating worse success rate in Scenario B. It could be useful as a collision-aware IK solver but it requires additional high-level planning for more complex scenarios, as demonstrated in the *multimedia attachment*.

3) MPPI: While the authors of [11] claim that the operating frequency of their MPPI implementation exceeds 100 Hz, they clarify that this is the case only for a static environment. Assuming that obstacles are not fixed, and since it is impossible to pre-compute and store the scene distances, then the distance evaluations must be repeated at each iteration. Since the sampling and rollouts are performed in joint space, the use of our learned function is justified. At each iteration, it is required to calculate distances between the environment and $N \cdot H$ robot configurations. The input for the learned function $\Gamma(q, y)$ is repeated S times, once for each spherical obstacle position; thus, a fall in performance is expected with the increase in the number of spherical obstacles. As a baseline to compare with, we again choose to approximate the robot body with spheres and evaluate collisions by calculating task-space distances between these spheres and spherical obstacles in the robot's workspace. Such baseline would seem computationally cheaper than NN evaluation; however, a highly parallelized forward pass of the neural network turns out to be faster. Tables V and VI show the comparison between the naive distance computation between obstacle spheres and robot geometry approximated with spheres. All experiments were conducted on a 12-core 3.7 GHz CPU and RTX3090 GPU. For the case when there are $S = 2$ obstacles in the workspace, our approach shows significantly faster iterations at 92 Hz versus 66 Hz. On average, distance computation using the learned function $\Gamma(q, y)$ is 40% faster than the baseline. Additionally, Fig. 5 demonstrates how this controller frequency scales with the value S. This is showcased in the *multimedia attachment* where we present multiple simulations of both scenarios with increasing obstacle speeds and adversarial behaviors.

4) Comparison: As shown in the evaluations reported in Tables V and VI and Fig. 5, both reactive control approaches perform well in terms of avoiding collisions using the learned Neural- JSDF. While QP-IK is faster and exhibits more reactive behavior, it may get stuck in local minima in the optimization, or fail to recover from an odd joint configuration caused by the instantaneous reactivity as shown for both scenarios in the *multimedia attachment.* On the other hand, while the MPC is slower than QP-IK, it manages to avoid the problematic regions where obstacles are moving very fast, escaping oscillatory

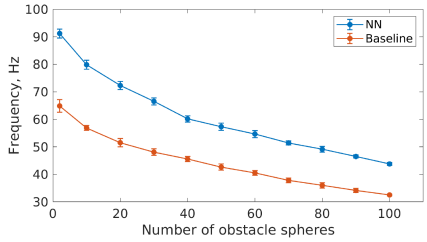

Fig. 5. The MPPI frequency (Hz) and variance as a function of number of spherical obstacles, S, in the workspace of the robot. The blue line is for collision checking via learned function and the red is for the baseline with spherical approximation of the robot. Values averaged on 10 runs.

reactive behaviors exhibited by the QP-IK and ultimately reaching the target.

5) Real Robot Experiment: We further validated the approach on a real Franka Emika Panda being controlled at 1 kHz by a custom low-level torque controller, similar to [11]. Obstacles and the goal are tracked with OptiTrack at a 120 Hz rate. We replicate scenarios A and B and stress test the system with adversarial obstacle motions as shown in the *multimedia attachment*. Fig. 4 demonstrates a successful reaching task while avoiding colliding with the human.

VII. Conclusion and Future Work

We have presented our method for learning the minimal distances between the robot and its environment as a neural joint space implicit distance function. This distance is a function of the robot joint state and the coordinates of a point in its workspace. We can efficiently compute distances between the robot in arbitrary configuration and obstacles represented as a set of spheres of various radii. The gradient of the learned function with respect to inputs can be treated as the repulsive vector in the joint space of the robot, allowing for collision avoidance constraint formulation in the QP Inverse Kinematics solver. While the learned network contains many parameters, due to high parallelization, it still outperforms the naive baseline in terms of controller frequency. This property allows efficient use of the learned function as a collision-checker in sampling-based MPC control.

While we have investigated scenarios where obstacles in the robot's workspace are approximated with spheres, this method can also be used with obstacles represented as point clouds. We leave corresponding implementations for future work. Another interesting direction would be to expand the use of repulsion gradients and apply them as a heuristic to guide the sampling to further improve the performance of the MPPI approach, similar to [21]. The distance-to-collision and cosine similarity between the sampled acceleration and repulsion direction could be used to introduce re-projected samples that navigate the robot around the obstacle.

REFERENCES

[1] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Robot. Autom.*, vol. 4, no. 2, pp. 193–203, Apr. 1988.

[2] N. Harrison, W. Liu, I. Jang, J. Carrasco, G. Herrmann, and N. Sykes, "A comparative study for obstacle avoidance inverse kinematics: Null-space based vs. optimisation-based," in *Towards Autonomous Robotic Systems*. A. Mohammad, X. Dong, and M. Russo, Eds., Cham, Switzerland: Springer International Publishing, 2020, pp. 147–158.

[3] S. Haddadin, A. De Luca, and A. Albu-Schäffer, "Robot collisions: A survey on detection, isolation, and identification," *IEEE Trans. Robot.*, vol. 33, no. 6, pp. 1292–1312, Dec. 2017.

[4] M. Khoramshahi, G. Henriks, A. Naef, S. S. M. Salehian, J. Kim, and A. Billard, "Arm-hand motion-force coordination for physical interactions with non-flat surfaces using dynamical systems: Toward compliant robotic massage," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 4724–4730.

[5] S. Li, N. Figueroa, A. Shah, and J. A. Shah, "Provably safe and efficient motion planning with uncertain human dynamics," *Robot.: Sci. Syst.*, 2021.

[6] M. Koptev, N. Figueroa, and A. Billard, "Real-time self-collision avoidance in joint space for humanoid robots," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 1240–1247, Apr. 2021. [Online]. Available: https://doi.org/10.1109/LRA.2021.3057024

[7] S. S. M. Salehian, N. Figueroa, and A. Billard, "A unified framework for coordinated multi-arm motion planning," *Int. J. Robot. Res.*, vol. 37, no. 10, pp. 1205–1232, 2018. [Online]. Available: https://doi.org/10.1177/0278364918765952

[8] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 4030–4035.

[9] S. Faraji and A. J. Ijspeert, "Singularity-tolerant inverse kinematics for bipedal robots: An efficient use of computational power to reduce energy consumption," *IEEE Robot. Automat. Lett.*, vol. 2, no. 2, pp. 1132–1139, Apr. 2017.

[10] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *J. Guid., Control, Dyn.*, vol. 40, no. 2, pp. 344–357, 2017. [Online]. Available: https://doi.org/10.2514/1.G001921

[11] M. Bhardwaj et al., "STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Proc. 5th Conf. Robot Learn.*, A. Faust, D. Hsu, and G. Neumann, Eds., vol. 164, Nov. 2022, pp. 750–759. [Online]. Available: https://openreview.net/forum?id=ceOmpjMhlyS

[12] A. Escande, S. Miossec, and A. Kheddar, "Continuous gradient proximity distance for humanoids free-collision optimized-postures," in *Proc. IEEE-RAS 7th Int. Conf. Humanoid Robots*, 2007, pp. 188–195.

[13] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick, "Real-time collision avoidance with whole body motion control for humanoid robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 2053–2058.

[14] C. Fang, A. Rocchi, E. M. Hoffman, N. G. Tsagarakis, and D. G. Caldwell, "Efficient self-collision avoidance based on focus of interest for humanoid robots," in *Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots*, 2015, pp. 1060–1066.

[15] M. Schwienbacher, T. Buschmann, S. Lohmeier, V. Favot, and H. Ulbrich, "Self-collision avoidance and angular momentum compensation for a biped humanoid robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 581–586.

[16] S. Zimmermann, M. Busenhart, S. Huber, R. Poranne, and S. Coros, "Differentiable collision avoidance using collision primitives," 2022. [Online]. Available: https://arxiv.org/abs/2204.09352

[17] P. Liu, K. Zhang, D. Tateo, S. Jauhri, J. Peters, and G. Chalvatzaki, "Regularized deep signed distance fields for reactive motion generation," 2022. [Online]. Available: https://arxiv.org/abs/2203.04739

[18] L. Montaut, Q. Le Lidec, V. Petrik, J. Sivic, and J. Carpentier, "Collision detection accelerated: An optimization perspective," in *Proc. Robot.: Sci. Syst.*, New York, NY, USA, Jun. 2022, doi: 10.15607/RSS.2022.XVIII.039.

[19] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1096–1114, Aug. 2020.

[20] Y. Zhi, N. Das, and M. Yip, "DIFFCO: Auto-differentiable proxy collision detection with multi-class labels for safety-aware trajectory optimization," 2021. [Online]. Available: https://arxiv.org/abs/2102.07413

[21] C. Kew, B. A. Ichter, M. Bandari, E. Lee, and A. Faust, "Neural collision clearance estimator for batched motion planning," in *Proc. 14th Int. Workshop Algorithmic Foundations Robot.*, 2020, pp. 73–89.

[22] D. Rakita, B. Mutlu, and M. Gleicher, "RelaxedIK: Real-time synthesis of accurate and feasible robot arm motion," in *Proc. Robot.: Sci. Syst. XIV*, 2018, pp. 26–30. [Online]. Available: http://www.roboticsproceedings.org/rss14/p43.html

[23] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 6010–6017.

[24] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[25] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Automat.*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[26] N. Vahrenkamp, T. Asfour, G. Metta, G. Sandini, and R. Dillmann, "Manipulability analysis," in *Proc. IEEE-RAS 12th Int. Conf. Humanoid Robots*, 2012, pp. 568–573.

[27] D. Frisch, "Distance between point and triangulated surface," Accessed: Jun. 3, 2022. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/52882-point2trimesh-distance-between-point-and-triangulated-surface

[28] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *Int. J. Robot. Res.*, vol. 35, no. 12, pp. 1477–1496, 2016.

[29] B. Mildenhall, P.P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 405–421.

[30] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, pp. 1–27, Mar. 2012.